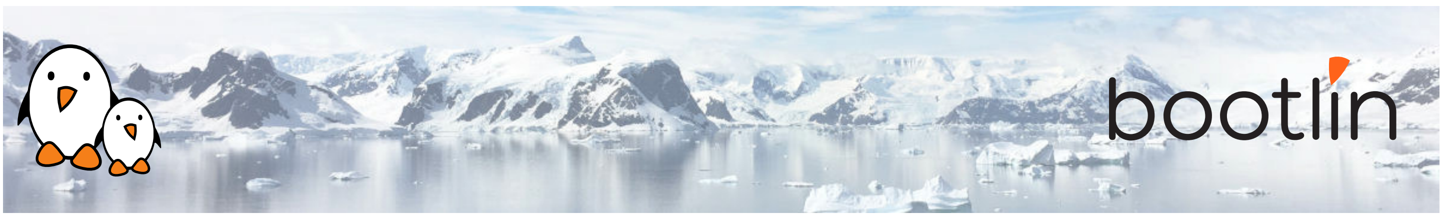
Linux debugging, profiling, tracing and performance analysis training

On-site training, 3 days
Latest update: November 30, 2021

Title	**Linux debugging, profiling, tracing and performance analysis training**
Training objectives	• Be able to understand why a system is loaded and what are the elements that contributes to this load using common Linux observability tools. • Be able to profile a complete userspace application and its interactions with the Linux kernel in order to fix bugs using standard tools. • Be able to analyze performance issues of an application (user/kernel) and understand how to address them. • Be able to analyze a kernel oops either live or post-mortem. • Be able to debug and trace a kernel module by using KGDB and kprobes. • Be able to profile a kernel module using perf, LTTNG and other tools.
Duration	**Three** days - 24 hours (8 hours per day).
Pedagogics	• Lectures delivered by the trainer: 50% of the duration • Practical labs done by participants: 50% of the duration • Electronic copies of presentations, lab instructions and data files. They are freely available at bootlin.com/doc/training/debugging.
Trainer	Clément Léger `https://bootlin.com/company/staff/clement-leger/`
Language	Oral lectures: English Materials: English.
Audience	Companies and engineers interested in debugging, profiling and tracing Linux systems and applications, to analyze and address performance or latency problems.

Prerequisites	• **Knowledge and practice of UNIX or GNU/Linux commands**: participants must be familiar with the Linux command line. Participants lacking experience on this topic should get trained by themselves, for example with our freely available on-line slides at bootlin.com/blog/command-line/. • **Minimal experience in embedded Linux development**: participants should have a minimal understanding of the architecture of embedded Linux systems: role of the Linux kernel vs. user-space, development of Linux user-space applications in C. Following Bootlin's *Embedded Linux* course at bootlin.com/training/embedded-linux/ allows to fulfill this pre-requisite. • **Minimal English language level: B1**, according to the *Common European Framework of References for Languages*, for our sessions in English. See bootlin.com/pub/training/cefr-grid.pdf for self-evaluation.
Required equipment	**For on-site sessions at our customer location, the customer must provide**: • Video projector • One PC computer on each desk (for one or two persons) with at least 8 GB of RAM, and Ubuntu Linux 20.04 installed in a **free partition of at least 30 GB** • Distributions others than Ubuntu Linux 20.04 are not supported, and using Linux in a virtual machine is not supported. • **Unfiltered and fast connection to Internet**: at least 50 Mbit/s of download bandwidth, and no filtering of web sites or protocols. • **PC computers with valuable data must be backed up** before being used in our sessions.
Certificate	Only the participants who have attended all training sessions, and who have scored over 50% of correct answers at the final evaluation will receive a training certificate from Bootlin.
Disabilities	Participants with disabilities who have special needs are invited to contact us at *training@bootlin.com* to discuss adaptations to the training course.



Hardware in practical labs

The hardware platform used for the practical labs of this training session is the **STMicroelectronics STM32MP157D-DK1 Discovery board** board, which features:

- STM32MP157D (dual Cortex-A7) CPU from STMicroelectronics
- USB powered
- 512 MB DDR3L RAM
- Gigabit Ethernet port
- 4 USB 2.0 host ports
- 1 USB-C OTG port
- 1 Micro SD slot
- On-board ST-LINK/V2-1 debugger
- Arduino Uno v3-compatible headers
- Audio codec
- Misc: buttons, LEDs

Day 1 - Morning

Lecture - Linux application stack

- Global picture: understanding the general architecture of a Linux system, overview of the major components.
- What is the difference between a process and a thread, how applications run concurrently.
- Userspace application memory layout (heap, stack, etc).
- MMU and memory management: physical/virtual address spaces.
- Kernel context switching and scheduling
- Kernel execution contexts: kernel threads, workqueues, interrupt, threaded interrupts, softirq

Lecture - Common observability tools

- Tools to use to monitor a Linux system: processes, memory usage and mapping, resources.
- Using *vmstat*, *iostat*, *ps*, *top*, *iotop*, *free* and understanding the metrics they provide.
- Pseudo filesystems: *procfs*, *sysfs* and *debugfs*.

Day 1 - Afternoon

Lab - Check what is running on a system and its load

- Observe running processes using *ps* and *top*.
- Check memory allocation and mapping with *procfs* and *pmap*.
- Monitor other resources usage using *iostat*, *vmstat* and *netstat*.

Lecture - Debugging an application

- Using *gdb* on a live process.
- Postmortem diagnostic using core files.
- Remote debugging with *gdbserver*.
- Extending *gdb* capabilities using python scripting

Lab - Solving an application crash

- Managing *gdb* from the command line, then from an IDE.
- Using *gdb* Python scripting capabilities.
- Debugging a crashed application using a coredump with *gdb*.

Day 2 - Morning

Lecture - Tracing an application

- Tracing system calls with strace.
- Tracing library calls with ltrace.

Lab – Debugging application issues

- Analyze dynamic library calls from an application using *ltrace*.
- Debug a misbehaving application using *strace*.

Lecture - Memory issues	Lab – Debugging memory issues
• Usual memory issues: buffer overflow, segmentation fault, memory leaks, heap-stack collision. • Memory corruption tooling, *valgrind*, *libefence*, etc. • heap profiling using *Massif*	• Buffer overflow investigation with *libefence*. • Memory leak and misbehavior detection with *valgrind* and *vgdb*. • Performance issues due to memory over allocation. • Visualizing application heap using *Massif*.

Day 2 - Afternoon

Lecture – Application profiling	Lab - Application profiling
• Performances issues. • Gathering profiling data with *perf*. • Analyzing an application callgraph using *Callgrind* and *KCachegrind*. • Filtering the data set. • Interpreting the data recorded by *perf*.	• Profiling an application with *Callgrind*/*KCachegrind*. • Analyzing application performances with *perf*. • Generating a flamegraph using *FlameGraph*.

Day 3 - Morning

Lecture - System wide profiling and tracing	Lab - System wide profiling and tracing
• System wide profiling using *perf*. • Using *kprobes* to hook on kernel code without recompiling. • *eBPF* tools (*bcctools*, *bpftrace*, etc) to trace complex scenarios. • Application kernel tracing and visualization using *kernelshark* or *LTTng*	• System profiling with *perf*. • IRQ latencies using *ftrace*. • Tracing specific kernel actions with *bpftrace*. • Tracing and visualizing system activity using *kernelshark* or *LTTng*

Day 3 - Afternoon

Lecture - Kernel debugging	**Lab - Kernel debugging**
• Understanding kernel *oops* messages. • Post mortem analysis using kernel crash dump with *crash*. • Memory issues (*KASAN*, *UBSAN*, *Kmemleak*). • Debugging the kernel using *KGDB* and *KDB*. • Kernel configuration options that are useful for debug.	• Analyzing an *oops* after using a faulty module. • Detecting undefined behavior with *UBSAN* in kernel code. • Find a module memory leak using *kmemleak*. • Debugging a module with *KGDB*.